NP-Completeness

- A language L is NP-complete if L belongs to NP and $L' \leq_{\mathrm{P}} L$ holds for all languages L' in NP
- Lemma 1: If L is NP-complete and belongs to P, then $\mathrm{P} = \mathrm{NP}$
  - Assume that L is NP-complete and belongs to P
  - It suffices to prove that every language in NP belongs to P
  - Let L' be a language in NP
  - Since $L' \leq_{\mathrm{P}} L$ and L belongs to P, we deduce that L' belongs to P

Transitivity of Polynomial-Time Reductions

- Lemma 2: If $L \leq_{\mathrm{P}} L'$ and $L' \leq_{\mathrm{P}} L''$, then $L \leq_{\mathrm{P}} L''$
  - Since $L \leq_{\mathrm{P}} L'$, there exists a constant $c_1 > 0$ such that we can decide L in $O(n^{c_1})$ time given a black box B_1 for deciding L'
    - The black box B_1 is called $O(n^{c_1})$ times, and each input string provided to B_1 is of length $O(n^{c_1})$
  - Since $L' \leq_{\mathrm{P}} L''$, there exists a constant $c_2 > 0$ such that we can implement any length-s call to B_1 in $O(s^{c_2})$ time given a black box B_2 for deciding L''
    - The black box B_2 is called $O(s^{c_2})$ times, and each input string provided to B_2 is of length $O(s^{c_2})$
  - Thus we can decide L in $O(n^{c_1} n^{c_1 c_2}) = O(n^{c_1(c_2+1)})$ time given a black box for deciding L''

Proving Problems NP-Complete

- Theorem 1: If L belongs to NP and there exists an NP-complete language L' such that $L' \leq_{\mathrm{P}} L$, then L is NP-complete
  - Assume that L belongs to NP and there exists an NP-complete language L' such that $L' \leq_{\mathrm{P}} L$
  - Let L'' belong to NP
  - Since L' is NP-complete, we have $L'' \leq_{\mathrm{P}} L'$
  - Since $L'' \leq_{\mathrm{P}} L'$ and $L' \leq_{\mathrm{P}} L$, Lemma 2 implies that $L'' \leq_{\mathrm{P}} L$

Establishing Existence of an NP-Complete Language

- Today, many thousands of languages (decision problems) are known to be NP-complete
- In virtually all cases, Theorem 1 is used to establish the NP-completeness of a given language
- However, in order to apply Theorem 1, we need to know that some other language is NP-complete
- How can we prove the existence of a "first" NP-complete language L
  - We need to prove that $L' \leq_{\mathrm{P}} L$ holds for all languages L' in NP
  - This seems like a daunting task, since the number of such languages L' is infinite

The Cook-Levin Theorem

- Theorem: SAT is NP-complete
  - In what follows, we present a high-level proof sketch
  - Let L be a language in NP
  - We need to prove that $L \leq_{\mathrm{P}}$ SAT
  - Since L is in NP, there is a polynomial-time verifier $\mathcal{A}(x, y)$
    - Let $p(n)$ denote a polynomial upper bound on the running time of $\mathcal{A}(x, y)$ when $|x| = n$
  - We will exhibit a polynomial-time transformation from any input string x to a propositional formula f such that f is satisfiable if and only if x belongs to L

The Variables of f

- There are three kinds of (boolean) variables in f
  - For any cell i, symbol j, and step k, there is a variable $T_{i,j,k}$ that is intended to indicate whether cell i contains symbol j at step k
  - For any cell i and step k, there is a variable $H_{i,k}$ that is intended to indicate whether the read-write head is scanning cell i at step k
  - For any state q and step k, there is a variable $Q_{q,k}$ that is intended to indicate whether the machine is in state q at step k
- There are $O(p(n)^2)$ variables of the first two types, and $O(p(n))$ variables of the third type

The Plan

- Suppose x belongs to L
- Thus there exists a short certificate such that $\mathcal{A}(x, y)$ outputs 1
- We will construct f as the conjunction of polynomially many (polynomial-size) subformulas
- Each of the subformulas enforces some aspect of our intended interpretation of the variables
- For any short certificate y such that $\mathcal{A}(x, y)$ outputs 1, there is exactly one satisfying assignment σ for f in which the certificate-related $T_{i,j,0}$ variables encode the certificate y
  - Under assignment σ, every variable gets assigned the truth value associated with its intended interpretation under execution $\mathcal{A}(x, y)$

The Plan (cont'd)

- Suppose x does not belong to L
- Thus $\mathcal{A}(x, y)$ outputs 0 for all y
- In this case, we will argue that our formula f is not satisfiable

Subformulas Enforcing A Basic Property of the $T_{i,j,k}$'s

- For any cell i and step k, we would like to ensure that exactly one of the $T_{i,j,k}$'s is true
  - We can use a disjunction over j to ensure that at least one of the $T_{i,j,k}$'s is true
  - For any distinct symbols j and j', we introduce the subformula $\neg T_{i,j,k} \vee \neg T_{i,j',k}$ to ensure that at most one of the $T_{i,j,k}$'s is true
  - The total size of the subformulas for a given i and k is $O(1)$
- The total size of all of these subformulas is $O(p(n)^2)$

Subformulas Enforcing a Basic Property of the $H_{i,k}$'s

- For any step k, we can add $O(p(n)^2)$ subformulas of size $O(1)$ to enforce that at most one of the $H_{i,k}$'s is true
  - Remark: We will not need to add a subformula to ensure that at least one of the $H_{i,k}$'s is true, as this fact will follow from other subformulas
- The total size of these subformulas over all k is $O(p(n)^3)$

Subformulas Enforcing A Basic Property of the $Q_{q,k}$'s

- For any step k, we can add $O(1)$ subformulas of size $O(1)$ to enforce that at most one of the $Q_{q,k}$'s is true
  - Remark: We will not need to add a subformula to ensure that at least one of the $Q_{q,k}$'s is true, as this fact will follow from other subformulas
- The total size of these subformulas over all k is $O(p(n))$

Subformulas Related to the Initial State

- We can enforce that at step 0, the tape contents properly encode an input pair with first component x
  - It is easy to enforce that the $T_{i,j,0}$ variables associated with the first component of the input pair encode the input string x
  - It is easy to enforce that the portion of the tape that follows the encoding of the input pair contains blanks
  - It is easy to enforce that a specific separator symbol appears between the first and second components
  - Altogether this requires $O(p(n))$ constant-sized subformulas
- We can enforce that the head is initially scanning cell 0
- We can enforce that the initial state is the start state

Subformulas Related to the Final State

- Assume our output convention for Turing machines requires that the following conditions are satisfied at step $p(n)$ in order to properly produce an output of 1
  - The tape is filled with blanks except that cell 0 contains the symbol 1
  - The head is scanning cell 0
  - The machine is in the halt state
- It is easy to introduce subformulas to enforce these conditions

The Remaining Subformulas

- Each of the subformulas discussed so far is simple in that it involves only one of the three kinds of variables
- We can get by with two more groups of subformulas
- One of these groups involves only the $T_{i,j,k}$'s and the $H_{i,k}$'s, and enforces that the contents of tape cell i cannot change at step k unless the read-write head is scanning cell i at step k
  - For any cell i, step k, and distinct symbols j and j', we add the subformula $T_{i,j,k} \wedge T_{i,j',k+1} \rightarrow H_{i,k}$
- The final group of subformulas is in some sense the most interesting, as it enforces the transition function of $\mathcal{A}$

Enforcing the Transition Function

- For any cell i, symbol j, step k, and state q, we add the subformula

$$(H_{i,k} \land Q_{q,k} \land T_{i,j,k}) \to (H_{i+d,k+1} \land Q_{q',k+1} \land T_{i,j',k+1})$$

  where d in $\{-1, 0, 1\}$, state q', and symbol j' are determined by the transition function of $\mathcal{A}$
- The total size of all of these subformulas is $O(p(n)^2)$

Proof Sketch of the Cook-Levin Theorem: Wrap-Up

- We claim that f is satisfiable if and only if there is a setting for the $T_{i,j,0}$ variables encoding the second argument y such that $\mathcal{A}(x, y)$ outputs 1
  - Hopefully the high-level intuition underlying this claim is clear
  - The formal proof is straightforward, but tedious
- Accordingly, f is satisfiable if and only if x belongs to L
- The transformation of x to f is easy to carry out in polynomial time
- Thus $L \leq_{\mathrm{P}}$ SAT, as required

3-SAT is NP-complete

- Earlier we saw that SAT $\leq_P$ 3-SAT
- By the Cook-Levin theorem, SAT is NP-complete
- Thus Theorem 1 implies that 3-SAT is NP-complete

Integer Linear Programming

- Integer linear programming
  - As in the case of linear programming, we are given an $m \times n$ matrix A, an $m \times 1$ column vector b, and $n \times 1$ column vector c, and we wish to find an $n \times 1$ column vector x such that $Ax \leq b$, $x \geq 0$, and $c^{\intercal}x$ is maximized
  - But now, we require all of the components of x to be integers
- Let ILP denote the decision version of the integer linear programming problem
- Let 0-1 ILP denote the restricted version of ILP in which all of the components of x are required to be 0 or 1
- We claim that 0-1 ILP is NP-complete

0-1 ILP is NP-Complete

- It is easy to argue that 0-1 ILP belongs to NP
- We claim that 3-SAT $\leq_P$ 0-1 ILP
  - Let f be a given 3-CNF formula
  - How can we transform (in polynomial time) f into a 0-1 ILP instance I such that f is satisfiable if and only if I is a positive instance of 0-1 ILP?

Maximum Independent Set

- Let $G = (V, E)$ be a given (undirected) graph
- A subset U of V is "independent" if no two vertices in U are connected by an edge
- The maximum independent set problem asks us to find a maximum-cardinality independent set of G
- In the decision version, denoted IS, we are given G and a bound k and we are asked to determine whether G contains an independent set of size at least k
- We claim that IS is NP-complete

IS is NP-Complete

- It is easy to argue that IS belongs to NP
- We claim that 3-SAT $\leq_P$ IS
  - Let f be a given 3-CNF formula
  - How can we transform (in polynomial time) f into an IS instance (G, k) such that f is satisfiable if and only if G has an independent set of size at least k?